Exception Handling in Java

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

What is Exception in Java

**Dictionary Meaning:** Exception is an abnormal condition.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

## What is Exception Handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

### **Advantage of Exception Handling**

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

## Hierarchy of Java Exception classes

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:

**Types of Java Exceptions**

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error



## Difference between Checked and Unchecked Exceptions

### **1) Checked Exception**

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

### **2) Unchecked Exception**

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### **3) Error**

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

## Java Exception Keywords

There are 5 keywords which are used in handling exceptions in Java.

Try - The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.

Catch -The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.

Finally - The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.

Throw -The "throw" keyword is used to throw an exception.

Throws -The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Public class JavaExceptionExample{

public static void main(String args[]){

try{

int data=100/0;

}catch(ArithmeticException e){System.out.println(e);}

System.out.println("rest of the code...");

}

}

o/p

Exception in thread main java.lang.ArithmeticException:/ by zero

rest of the code...

## Common Scenarios of Java Exceptions

There are given some scenarios where unchecked exceptions may occur. They are as follows:

### **1) A scenario where ArithmeticException occurs**

If we divide any number by zero, there occurs an ArithmeticException.

1. **int** a=50/0;//ArithmeticException

### **2) A scenario where NullPointerException occurs**

If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

1. String s=**null**;
2. System.out.println(s.length());//NullPointerException

### **3) A scenario where NumberFormatException occurs**

The wrong formatting of any value may occur NumberFormatException. Suppose I have a string variable that has characters, converting this variable into digit will occur NumberFormatException.

1. String s="abc";
2. **int** i=Integer.parseInt(s);//NumberFormatException

### **4) A scenario where ArrayIndexOutOfBoundsException occurs**

If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException as shown below:

1. **int** a[]=**new** **int**[5];
2. a[10]=50; //ArrayIndexOutOfBoundsException

A try block can be followed by one or more catch blocks.

* At a time only one exception occurs and at a time only one catch block is executed.
* All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

MULTIPLE CATCH

**public** **class** MultipleCatchBlock1 {

**public** **static** **void** main(String[] args) {

**try**{

**int** a[]=**new** **int**[5];

                a[5]=30/0;

               }

**catch**(ArithmeticException e)

                  {

                   System.out.println("Arithmetic Exception occurs");

                  }

**catch**(ArrayIndexOutOfBoundsException e)

                  {

                   System.out.println("ArrayIndexOutOfBounds Exception occurs");

                  }

**catch**(Exception e)

                  {

                   System.out.println("Parent Exception occurs");

                  }

               System.out.println("rest of the code");

    }

}

MULTIPLE TRY

**class** Excep6{

**public** **static** **void** main(String args[]){

**try**{

**try**{

     System.out.println("going to divide");

**int** b =39/0;

    }**catch**(ArithmeticException e){System.out.println(e);}

**try**{

**int** a[]=**new** **int**[5];

    a[5]=4;

    }**catch**(ArrayIndexOutOfBoundsException e){System.out.println(e);}

 System.out.println("other statement);

}**catch**(Exception e){System.out.println("handeled");}

   System.out.println("normal flow..");

 }

}

**Java finally block** is a block that is used to execute important code such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block.

**class** TestFinallyBlock{

**public** **static** **void** main(String args[]){

**try**{

**int** data=25/5;

   System.out.println(data);

  }

**catch**(NullPointerException e){System.out.println(e);}

**finally**{System.out.println("finally block is always executed");}

  System.out.println("rest of the code...");

  }

}

## Java throw keyword

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

**throw** exception;

**throw** **new** IOException("sorry device error);

**public** **class** TestThrow1{

**static** **void** validate(**int** age){

**if**(age<18)

**throw** **new** ArithmeticException("not valid");

**else**

      System.out.println("welcome to vote");

   }

**public** **static** **void** main(String args[]){

      validate(13);

      System.out.println("rest of the code...");

  }

}

# Java throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

**import** java.io.IOException;

**class** Testthrows1{

**void** m()**throws** IOException{

**throw** **new** IOException("device error");//checked exception

  }

**void** n()**throws** IOException{

    m();

  }

**void** p(){

**try**{

    n();

   }**catch**(Exception e){System.out.println("exception handled");}

  }

**public** **static** **void** main(String args[]){

   Testthrows1 obj=**new** Testthrows1();

   obj.p();

   System.out.println("normal flow...");

  }

}